CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

APPLIED PCs

by Fred Eady

Mission Possible: Achieve Cheap USB Connectivity

Exorbitant USB licensing fees and the high price of analysis tools have denied many of you access to the USB marketplace. A champion of the individual designer, Fred is on a mission to prove that it's possible to achieve personal USB connectivity without breaking the bank.

I just received a PICkit 1 Flash starter kit, and, to my surprise, the programming interface is USB! Imagine that—a USB port on the most basic of Microchip's development boards (see Photo 1). Although I think the PICkit 1's USB programming port is a good thing, I still have some problems with USB when it comes to pulling a personal USB project together.

Thanks to folks like Microchip, Cypress, and National Semiconductor, USB hardware is relatively cheap and easy to obtain. All of the companies go out of their way to provide useful example code, and some even offer comprehensive USB tutorials aimed at their products. On the other hand, if you want to market a USB-equipped product, you have to either fork out $2500 per year to join the club (i.e., USB Implementers Forum) or obtain a USB vender ID (good for two years) for a measly $1500. Either way, your product must pass various tests to be certified. When the words "test" and "certification" are used, it usually means more money out of your pocket that has to be offset by raising the product's market price.

You can't get something for nothing, and I'm sure the USB license fees are used to enhance the processes and tools implemented by the USB development community. It looks like the proceeds are being put to good use, because the free USB tools on the official USB web site are useful for preparing a product for USB certification.

USB license fees are small change to large companies. Unfortunately, $2500 or even $1500 may prevent a smaller enterprise from entering the USB market, because the license fee is just a small part of what is needed to seriously develop USB devices.

I had wanted to show you some of the devices, so I contacted a well-known producer of USB analyzers. The company's least expensive analysis tool runs for approximately $8000, and its top-of-the-line USB analyzers top out at more than $30,000. There are several negative words I could use to describe our conversations. Anyway, as you read this article, you won't find any of the company's equipment pictured or mentioned. The bottom line is that the company's actions indicated that it isn't interested in showing you its products.

So, I've decided to prove that you can obtain personal USB connectivity without spending tens of thousands of dollars on license fees, lab certifications, USB vendor logos, and expensive USB analysis tools. I am officially on a mission.

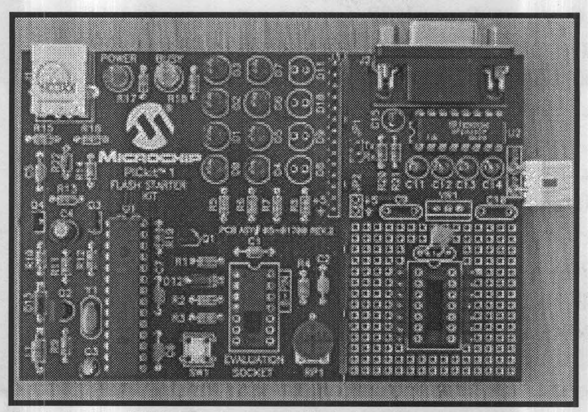

Photo 1—*The PICkit 1 Flash starter kit is designed to program and read the new 14-pin flash memory PICs and the legacy 8-pin flash memory parts. The PICkit 1 comes with an extensive software and firmware library that includes source code for the host and PIC USB interface. I populated the snap-off part of my PICkit 1 with a Sipex SP232ACP and supporting components.*

TRACKING DOWN USB

After I had decided to write this article, I set out to find a suitable and inexpensive USB analysis tool. To do so, I started my e-mail engine and donned my telephone headset. In less than a day, I located and obtained the holy grail of USB, an Ellisys USB Tracker 110 (see Photo 2).

The USB Tracker 110 is a small hardware device that traps, decodes, and displays a USB datastream flowing between the USB device being tested and a PC. Installing the USB Tracker 110 was a snap: I downloaded the latest version of the analysis software, UsbShow (www.usbtracker.com), and after a few mouse clicks, the USB Tracker 110 software and driver set were installed.

The next step involved connecting the USB Tracker 110 to the analysis computer. After the standard "I have found a new USB device" Windows message had appeared, I manually directed the installation wizard to install the USB Tracker 110 drivers that I had previously downloaded. I got a magic wand

from the Windows installation wizard, and the USB Tracker didn't smoke, so all was going well.

This is a good place to pause the USB Tracker 110 discussion and comment on the USB hardware I collected for analysis. I have two USB development boards and a PICkit 1 that can connect to my newly acquired USB Tracker 110. Let's begin by looking at ME Labs's PICProto USB development board.

PICProto USB

The reader response to my column titled "A P89C668 Development Board for 8051 Fans" was enough to tell me that the BASIC programming language is—as it relates to the 8051, PIC, and AVR—alive and well (*Circuit Cellar* 151). ME Labs offers a good PIC BASIC compiler called PicBasic. I got a copy of PicBasic Pro because it's more than a decent BASIC compiler. The professional version is relatively inexpensive, and it supports Microchip's flavor of low-speed USB, which is driven by the PIC16C745 and PIC16C765 microcontrollers.

The PICProto USB development board was refreshing because I had to build it completely from scratch. I had to fly in two components, a 6-MHz ceramic resonator and a series-B USB connector. Otherwise, the through-hole assembly process was quick and easy. A schematic and bill of materials are included with the bare silk-screened PICProto USB PCB.

For a complex interface, the USB hardware is just as simple as basic RS-232 hardware. In fact, I'll go out on a limb and say that a PIC-based USB hardware interface is actually simpler than a PIC-based RS-232 hardware interface.

The PIC16C745 and PIC16C765 USB microcontrollers are self-contained and don't require any of the auxiliary level-shifting circuitry that is common for true RS-232 implementations. Using the PIC16C765 or PIC16C745, a bare-bones USB hardware interface consists of the PIC, a couple of 0.1-µF bypass capacitors, a VUSB filter capacitor, a ceramic resonator, a resistor, and the series-B USB connector. Even though the PIC USB solution provides for a simpler hardware interface, USB communication requires much more effort to implement.

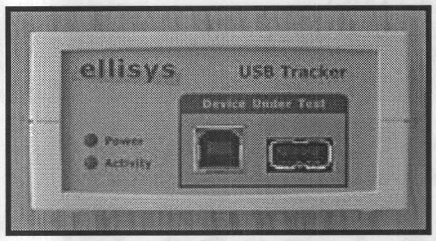

Photo 2—*You don't need anything but this little box, a downloadable software application, and three USB cables to help unlock the mysteries of USB. It costs less than $900.*

On the PICProto USB development board, the 28-pin PIC16C745 IC socket lies inside the 40-pin PIC16C765 IC socket. I wanted to be able to mount either the 40-pin PIC16C765 or the 28-pin PIC16C745 on the PICProto USB development board. So, I used machined header pins to populate the 40-pin socket pads instead of a standard 40-pin socket. This allowed me to solder a standard 0.3″ 28-pin socket inside the 40-pin socket footprint. My completed PIC-less PICProto USB development board is shown in Photo 3.

Because there is no "F" in their names, the PIC16C745 and PIC16C765 microcontrollers are available as either windowed ultraviolet erasable parts or one-time programmable (OTP) parts. I have a fancy timer-equipped EEPROM eraser, but I've been spoiled by the easy-to-program, flash memory-based PICs. Because the new generation of flash memory-based USB PICs wasn't available when I started this article, I used the MPLAB ICE 2000 and a PCM16XQ1 processor module to stand in for the windowed PIC16C745 and PIC16C765.

Before attempting to read the PICProto USB development board's USB datastream with the USB Tracker 110, it may be a good idea to check to see if all of the solder joints took. A small USB demo program that moves the test computer's cursor is included with the PicBasic Pro compiler. I loaded that puppy into the MPLAB ICE 2000 to see if I could twirl the cursor.

Using the MPLAB ICE 2000 instead of the real thing required that I invoke the MPLAB IDE. Normally, that would have meant loading and running a separate IDE for the PicBasic Pro compiler. Not in this case. The PicBasic Pro compiler is capable of running as a language toolset within the latest version of the MPLAB IDE. Although being able to run PicBasic Pro and MPLAB in a single IDE is a good thing in terms of development, there is another upside to this union: PicBasic Pro generates a standard .cod file that allows for debugging using the MPLAB ICE 2000 hardware.

After plugging in the MPLAB ICE 2000 PCM16XQ1 USB processor module and attaching a 40-pin DIP device attachment module to the end of the processor module's cable, I carefully plugged the MPLAB ICE 2000 device attachment's gold-plated 40-pin DIP header into the 40-pin header socket that I had installed on the PICProto USB development board. Then, I jumpered the PICProto USB development board for USB-supplied power.

At that point, I loaded the PicBasic Pro USB demo, USBMOUSE.BAS, in the MPLAB IDE. I couldn't get a good reset on the MPLAB ICE 2000. After clicking the MPLAB IDE Run icon a few times without success, I figured that something wasn't working correctly.

I first took the software problem determination route. (After all, my soldering should be perfect.) I muddled around, trying this and that with files and such with no joy. OK, maybe I did have a problem with the PICProto USB development board hardware. So, I disconnected the PICProto USB development board and took it to the bench for a look under the magnifier. As I had expected, all was well with the soldering job and component placement.

I did not cut any of the default

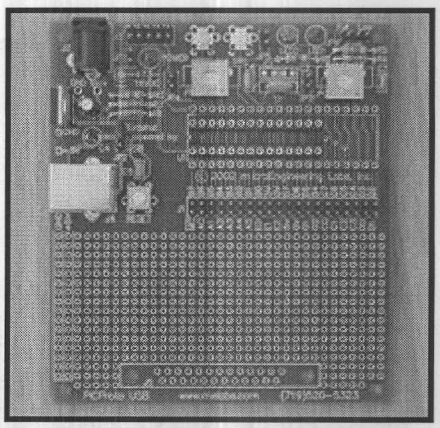

Photo 3—*All of the goodies that complement the everyday PIC are included with this board. There are a couple of potentiometers, a pair of LEDs, and two push-button switches. The 25-pin connector pad layout suggests that a serial-to-USB thing could happen in the prototype area.*

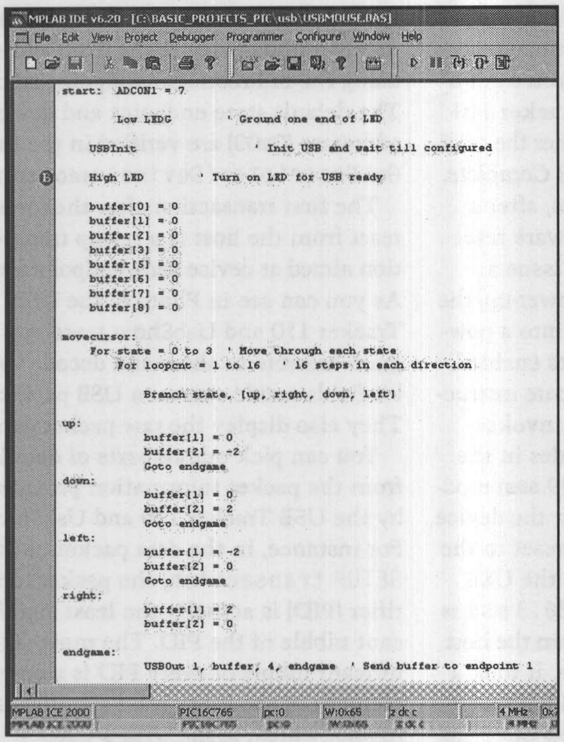

Photo 4—*I wanted to show you the* `USBInit` *instruction and the MPLAB ICE 2000 breakpoint. Now you should have an idea of how PicBasic Pro fits inside an MPLAB IDE session. Note that only two PicBasic Pro USB commands are used, USBInit and USBOut.*

jumpers on the PICProto USB development board. The only live jumper pins were the power-source pins. I decided to reattach the MPLAB ICE 2000 and move the powered-by jumper from USB to external. The MPLAB ICE 2000 reset was successful, and I was able to configure the MPLAB ICE 2000 to use the PICProto USB development board's power and clock. Obviously, the emulator and associated electronics drew a bit more current than the USB was willing to supply at that point. Despite the little drawback, it was good.

I had already created a project directory and copied the USBMOUSE.BAS file into it. A study of the USB documentation that was included with the PicBasic Pro compiler indicated that I would need to add supporting files to the project as well. These files, which were included with the compiler, provide a basis for the USB hardware layer that is intended to reside in the PIC firmware. The PicBasic Pro USB support files are based on the original Microchip USB support files and have been modified to compile under PicBasic Pro.

Having put my emulator hardware problem behind me, I was on my way to compiling the USB demo code and controlling a cursor. Well, not quite on my way. I clicked the MPLAB IDE's Build icon and was greeted with what seemed to be a million error messages, which wasn't good.

My first clue was the first error, which stated that it could not open a particular include file, P16C765.INC. That's easy enough, I said to myself. I'll just rename the PicBasic Pro 16C765.INC to P16C765.INC and things will once again be good. Ha! My mouth was still open when the dust stirred by a million more error messages had settled. OK, maybe the list file would reveal an answer. Duh. My second clue was on the first error line of the listing. `MPASM header` was the comment beside the lost P16C765.INC file.

You can configure the PicBasic Pro to use either the MPLAB assembler (MPASM) or the internal PicBasic Pro assembler (PM). Well, there's a big check mark in the "Use MPASM Assembler" box under the MPLAB IDE's project build options. I had renamed the PM include file (16C765.INC) to fool the MPLAB IDE assembler, but the MPLAB IDE assembler didn't bite on the contents of the newly monikered file.

I checked my Win2K path variable and indeed the entry for the Microchip MPLAB IDE include files was there. Without question (I was desperate), I simply copied the Microchip-provided MPLAB IDE P16C765.INC and P16C764.INC include files into my PicBasic Pro USB project directory.

Here we go. After clicking on the MPLAB IDE Build icon, I was humming my favorite Bob Marley tune, "Jammin'." After another click on the MPLAB IDE Run button, I was growing dreadlocks. The test machine's cursor was going in circles and dancing to the reggae beat. It was good indeed.

BACK ON TRACK(ER)

If you're a drag racing fan, you know that most of the real work is done in the pits before the race. The same can be said for working with USB. Before powering up the PICProto USB development board, I reread Jan Axelson's book, *USB Complete: Everything You Need to Develop Custom USB Peripherals*. [1] In addition, I perused the Microchip USB documentation and datasheets to get the particulars on the PIC16C765 and PIC16C745 USB microcontrollers. A good collection of USB

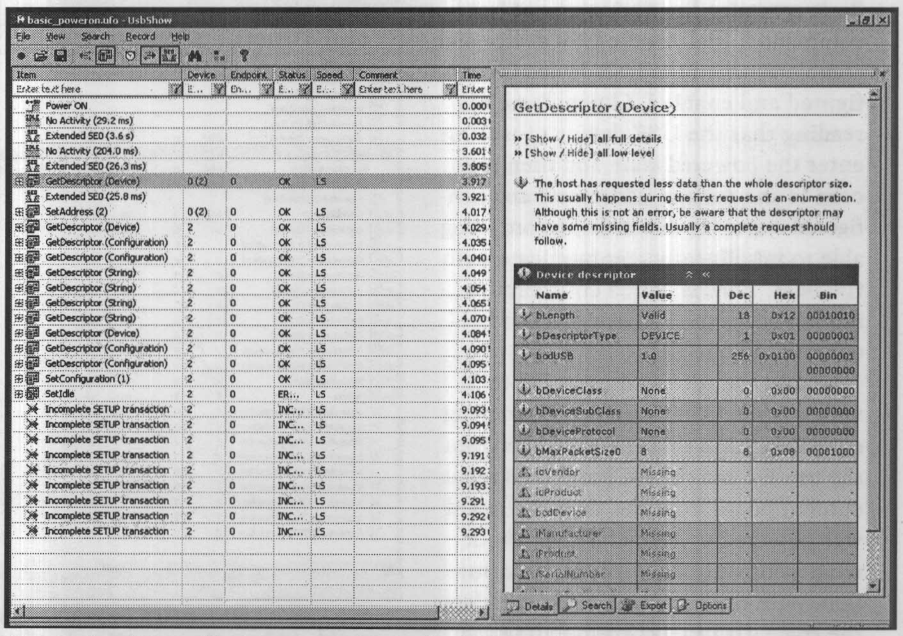

Photo 5—*I couldn't possibly show you everything, so I've decided to give you copies of all of my traces. You can view them using the USB Tracker 110 display software, UsbShow.*

data comes with Microchip's PICDEM USB development board.

After working out the initial emulator and compiler bugs, I decided to test run the USB Tracker 110. I used the resulting USB trace in conjunction with Axelson's book and the USB specification to determine what was needed from a firmware standpoint to be successful with USB. Now, I'll explain how it went.

Slow-motion photography is often used to study the minute details of a fast-moving event. You can do a similar thing electronically using an emulator like the MPLAB ICE 2000. So, I added a breakpoint to the PicBasic Pro compiler demo USB code to see if I could stop the USB process and examine it up to that point (see Photo 4). The PicBasic Pro compiler only supports three USB BASIC commands: `USBINIT`, `USBIN`, and `USBOUT`. According to Axelson's book and the USB specifications, three simple USB commands won't cut it. However, I saw two of those simple little commands wiggle a cursor using a rudimentary PIC circuit and USB. It was time to sip from the Holy Grail.

I inserted the USB Tracker 110 into the loop with the analysis computer at the USB Tracker 110 analyzer tap. The test computer and PICProto USB development board were plugged into the USB Tracker 110's device under test sockets. According to the PicBasic Pro Basic compiler description, the `USBINIT` command initializes the USB device and completes when the USB device is configured and enabled. I knew from my reading that the USB device must first enter the powered state and then proceed to the default, addressed, and configured states (in that order) before being able to intelligently communicate with the host. That is called enumeration. But how does the PicBasic Pro `USBINIT` command do it all?

Behold the screen shot in Photo 5, which is the USB Tracker 110's view of everything USB that transpired between the test PC and the PICProto USB development board from the time the board was powered up. Wow! Think about what you could do with this information. With the trace, you could correlate the trace data to a corresponding segment of PicBasic Pro source code, and you should be able to investigate the particulars of the `USBINIT` command. Let's work through an example. You can follow along using the USB Tracker 110 trace data in Photo 5 and either the original USB specification or *USB Complete.*

According to both sources, after a successful initial USB hardware reset sequence, the host PC will issue a GetDescriptor request. At power-up, the PIC-based USB device goes into a powered state with its interrupts enabled. When the PIC is able to execute instructions, the USBINIT macro invokes the `InitUSB` code that resides in the Microchip-supplied USB_CH9.asm module. When the host sees that the device is powered, it issues a USB reset to the powered device. Looking at the USB trace, the `Extended SE0 (26.3 ms)` is most likely the USB reset from the host. After starting the USB trace, it took a few seconds to plug in the PICProto USB development board's USB cable and start the MPLAB ICE 2000 emulator.

The host USB reset then triggers the PIC's USB reset interrupt, which configures the PIC's USB address to 0x00 and enables endpoint zero. This collection of zeros is in the default state because every USB device must have an active endpoint zero at that point. In the default state, a USB address has not been assigned by the host, and the host expects to be able to talk to the newly found device at the 0x00 address using the bidirectional endpoint zero. The default state endpoint and device addresses (0x00) are verified in the first `GetDescriptor(Device)` trace entry.

The first transaction after the forced reset from the host is a set-up transaction aimed at device zero, endpoint zero. As you can see in Photo 6, the USB Tracker 110 and UsbShow trace program are able to show and decode the bit fields inside common USB packets. They also display the raw packet data.

You can pick out all sorts of details from the packet information provided by the USB Tracker 110 and UsbShow. For instance, in the data packet of the `SETUP transaction`, the packet identifier (PID) is actually the least significant nibble of the PID. The most significant nibble of every PID is a complement of the PID's least significant nibble, which, along with a CRC word, helps to ensure data integrity.

In the aforementioned example, the token packet is followed by a data packet, which contains the set-up request information. Axelson says that although the host asks for 64 bytes, it will only read the first 8 bytes because all it really wants is the eighth byte of the device descriptor, which contains the maximum packet size. She's right.

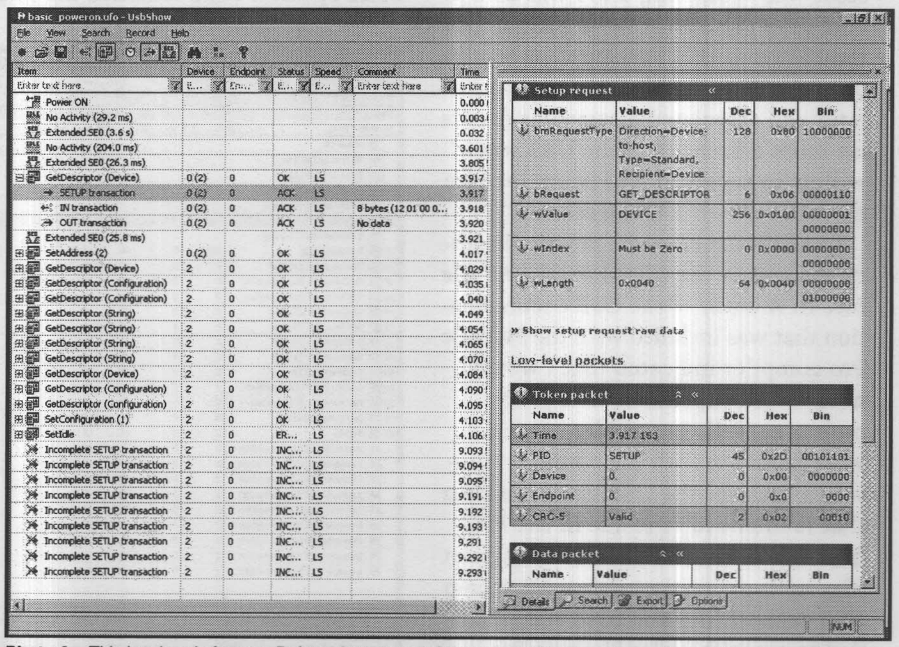

Photo 6—*This is a handy feature. Before obtaining a USB Tracker 110, I found myself searching through the pages of the USB specification and Microchip USB source code looking for the tables and declarations that defined the various bit fields.*

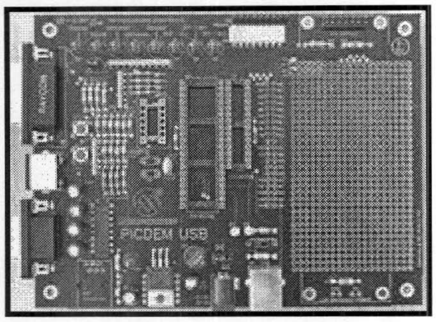

Photo 7—*If you prefer an assembled alternative to the PICProto USB, the Microchip PICDEM USB provides the basic functionality of the PICProto USB in addition to the ability to experiment with using USB to control an LCD and game pad. The PICDEM USB LEDs, which can be switched out with a jumper or at the firmware level, illuminate to signal each state of enumeration.*

Axelson's Carnac-inspired foresight is reinforced by an information message that UsbShow posts for the `SETUP transaction`. The UsbShow message tells you that the retrieval of less than 64 bytes is not an error and that the host would later ask again for the device descriptor information. Looking ahead in the trace shows you that the entire device descriptor is read in after the device address is established.

Using the '16C765 and '16C745 USB micros eliminates a great deal of USB configuration confusion because they're low-speed devices. Low-speed devices are limited in many ways, one of which is the maximum packet size, which is set at 8 bytes. You can see the 8-byte limit enforced throughout the USB trace.

Moving to the next transaction, the `IN transaction` pulls the 8 bytes of the 64 bytes of requested data from the PICProto USB development board's PIC firmware. The host has just enough information and status to grant the PICProto USB development board a USB address. But before knighting the development board, the host wisely resets it to make sure it is in the default state before sending the `SetAddress` request.

After the `SetAddress` request is acknowledged, the PICProto USB development board is considered to be in the addressed state. As you can see in the trace, the development board was reset and assigned an address (2). Remember that everything in USB is about the host. So, `IN` means to the host, and `OUT` is from the host.

From what you've seen thus far, you probably have a good idea of how the rest of the USB enumeration process will flow. Looking at the trace, you can see that device, configuration, and string descriptors are collected from the PICProto USB development board's firmware as the USB enumeration sequence progresses. The Windows OS assimilates all of the data collected from the PIC and finally sends a `SetConfiguration` request, which ultimately results in placing the development board in the configured state. After configuration, the development board can pass data to and receive data from the host PC (the test computer).

THE SONG REMAINS THE SAME

I unplugged the 40-pin MPLAB ICE 2000 device adapter from the PICProto USB development board and plugged it into the 40-pin PIC16C765 socket on my PIDCEM USB development board. I clicked on the MPLAB IDE Run icon and, lo and behold, the PICProto USB development board demo code ran on the PICDEM USB as it should.

Although cosmetically modified to compile inside PicBasic Pro, the core USB code and fundamental hardware design are identical. The PICDEM USB development board differs from the ME Labs PICProto USB development board only in the amount of on-board equipment. The former comes assembled with a USB CD-ROM containing support code and documentation, a preprogrammed PIC16C765 full of demo code, a 3′ USB A-B cable, a blank '16C745, and a blank '16C765 (see Photo 7). It also includes a serial port, a game port, a PS/2 keyboard/mouse port, enumeration status LEDs, and a backlit LCD landing pad.

I wasn't able to show you every detail of the USB traces, and I would need a few more pages to take a complete look at the features offered by the USB Tracker 110 and UsbShow. So, instead of leaving you hanging, I've included all the USB trace data that I took from the PICKit1 Flash starter kit, the PICProto USB, and the PICDEM USB. You may download the data from the *Circuit Cellar* ftp site. The core USB source code is on Microchip's web site.

If you're wondering how you're going to read the USB traces, simply download your freeware copy of UsbShow (www.usbtracker.com). The UsbShow software will display the USB traces I've provided and the detail contained within them. However, you won't be able to take your own USB traces without a USBTracker 110. The USB Tracker web site also has a great pictorial overview of what the USB Tracker 110 and UsbShow can do. Reviewing the overview will make it easier to interpret the USB traces. USB is complicated, but at least it's embedded. ■

Fred Eady has more than 20 years of experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications. Fred may be reached at fred@edtp.com.

PROJECT FILES

To download the UsbShow files, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2003/157.

RESOURCES

USB Tracker 110 distributors
Saelig Company
www.saelig.com

USB Wholesale
www.usbwholesale.com

REFERENCE

[1] J. Axelson, *USB Complete: Everything You Need to Develop Custom USB Peripherals*, Lakeview Research, Madison, WI, 2001.

SOURCES

USB Tracker 110
Ellisys Sarl
www.ellisys.com

MPLAB ICE 2000, PIC16C745, PIC16C765, PICDEM USB, and PICkit 1 starter kit
Microchip Technology, Inc.
www.microchip.com

PicBasic Pro compiler, PICProto USB
microEngineering Labs, Inc.
www.microengineeringlabs.com

SP232ACP Transceiver
Sipex Corp.
www.sipex.com